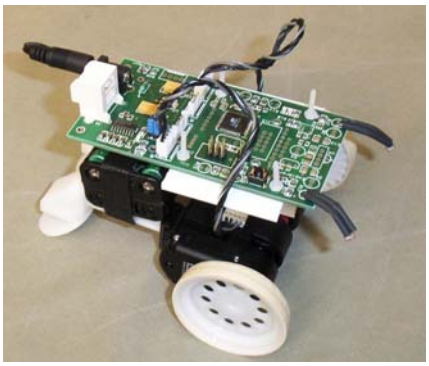
AI Motor Introductory Kit
User's Manual



AI Motor Introductory Kit
User's Manual

The AI Motor Introductory Kit was designed to familiarize you with AI Motors. By following the examples in this manual, you'll demonstrate and learn to use the key features unique to AI Motors. And you'll end up with a 2-wheeled robot programmable in C and extendable with additional sensors.

Designed specifically for robotics, AI Motors (AIMs) are building blocks combining mechanical connectors, servo-motor actuator, sensors, and smarts. All motors communicate via a single serial port, but at logic rather than RS232 voltage levels. Unique features include programmable servo and full-rotation modes, position feedback (position encoder and turns counter), and mechanical loading feedback. This kit combines 2 AI Motors with a PC serial interface, embedded microcontroller, and sensors. This allows you to initialize and manually control AIMs from a personal computer as well as build and program an autonomous mobile robot. All necessary software is included.

The introductory kit circuit board includes an RS232 serial port, LED displays, and a light sensor. Headers provide for digital input/output and analog input. The CPU is an ATmega16, with 16 Kbytes of FLASH program memory and 1 KByte of SRAM data memory. It can be programmed over the serial port – *a chip programmer or programming pod is not needed*. The controller board uses free and open-source software, including an integrated development environment with assembler, C compiler, and special-function library.

The target audience includes the following.

- **Hobbyists and Developers**. With the included libraries, most people familiar with C and embedded programming should find custom development of AI Motor applications quick and straightforward.
- **Beginning Hobbyists**. The software distribution includes function libraries and an easy to modify example. But you'll need to study C and embedded programming.

This manual is divided into three sections. The first, Quick Start, lists the kit contents then walks you through several examples using the AI Motors Tool PC software and the demos built into the controller board. The second section provides specifications for the controller board and summarizes specifications for AI Motors. The third section introduces programming the controller board using the free GCC compiler and libraries. This is a lengthy manual, but the Quick Start section will get you up and running AI Motors in just a few minutes.

Part 1: Quick Start

This quick start walks you through initializing and controlling AI Motors through a GUI tool and running a demo built in to the controller board. For additional information on AI Motors, see the AI-701 manual. More information on the controller board is included later in this document.

To use the software, you'll need a personal computer running

- WindowsXP preferred, Windows3.1/NT/95/98/2000 should be fine.
- At least 70 Mbytes free disk space (for full C compiler tool chain; only a few Mbytes for the AIM demo software)
- A cable for your PC's serial port. For most PCs this will be a DB9 female on one end and a RJ11-6P4C on the other. See Table 1.

To use the kit and assemble the 2-wheeled robot, you'll need the following.

1. a small flat-head screwdriver
2. a small Phillips-head screwdriver
3. pliers (needle nose preferred)
4. 6 AAA-size batteries. Rechargeable NiMH recommended.

A flashlight will make the light-seeking demo program more entertaining.

If you want to skip ahead to the robot demo, you can skip Example 2, but you'll still need to install the AI Motor Tool and go through Example 1 to properly initialize the motors.

1.1 Kit Contents

- 2 AI-701 motors (7 kg*cm stall torque), each with
  - 11 mechanical attachment pieces plus screws and nuts
  - 2 cables
  - replacement gears
- Megatrol-1 circuit board (partially populated) with RS232 level translator, ATmega16 microcontroller, and 2 light sensors.
- DB9 to RJ11 adapter with RJ11 cable
- battery holder with plug
- Velcro strip for securing holder with batteries
- 2 board mounting brackets
- 4 2-56 x ¾" nylon screws
- 4 2-56 x 3/8" nylon screws
- 12 4-40 nylon washers
- 2 3/16" spacers
- 2 rubber bands… er, traction treads

- CD-ROM containing
  - AI Motor Tool
  - Atmel Studio 4
  - GCC compiler and libraries
  - Source code for demo
  - AI-701 manual
  - This document

NOTE: *Don't substitute a higher voltage unregulated supply*. Most DC supplies are unregulated. A 7.5 V 1 A supply with the light loading of a single motor results in a roughly 10 V output. A 7.5 V 1 A supply is adequate for 4-6 AI Motors.

1.2 Using the AI Motor Tool

The AI Motor Tool, provided by the motor manufacturer, is a graphical user interface to used to initialize and manually control AI Motors. It isn't fully implemented, so we'll walk you around the blind spots.

1.2.1 Installing the AI Motor Tool

The AI Motor Tool is included on the kit's CD-ROM, and can be downloaded from www.Garage-Technologies.com. Go to the CD-ROM folder *Megarobotics software* and copy *AI Motor* Tool to your computer's desktop. Click on the .zip (compressed file) icon.

1.2.2 Example 1: Initializing the Motors

This example walks you step by step on how to use the GUI to initialize the motors for use in later examples. Figure 1 illustrates the connectors and jumper settings.

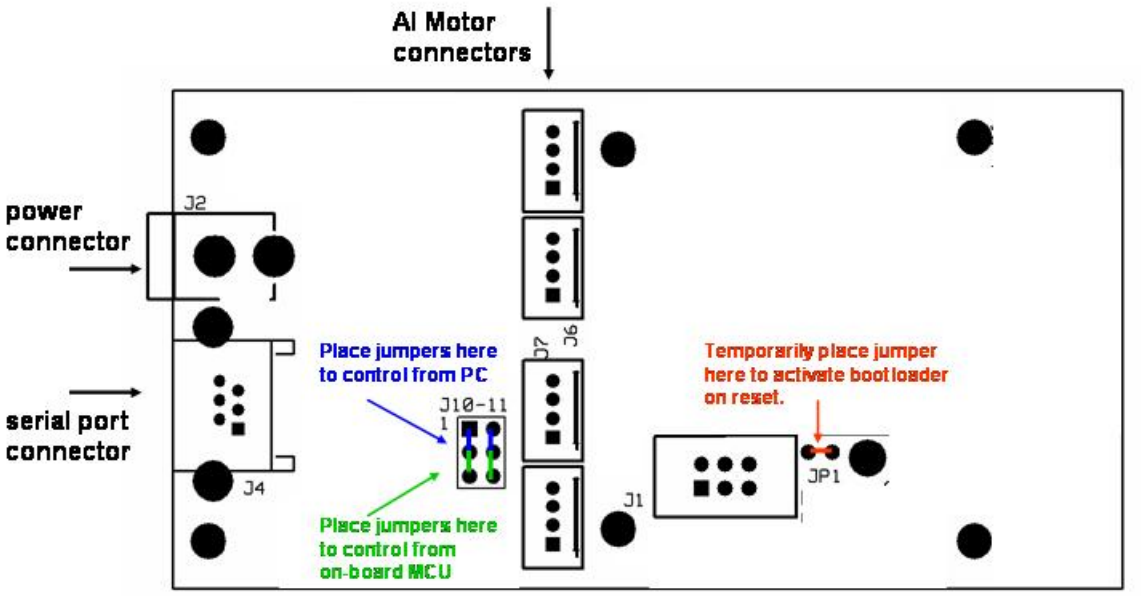


Figure 1 – Connectors and Jumpers

Note: Additional connectors for a Devantech SRF08 sonar sensor and power are not shown; these will be documented in an upcoming application note.

1. Connect the serial cable between your PC's serial port and the AIM controller board. Run the AI Motor Tool. Be sure the *SingleSetting* panel is selected.
2. The AI Motor Tool communicates via the PC's serial port. For most PCs this is Com 1, so select Com1 in the upper left of the GUI. AI Motors ship with a default ID of 0 and a default baud rate of 57,600, so set the tool's baud rate to 57600 (also in the upper left of the GUI).
3. Be sure the jumpers J10 and J11 are in the upper position ("control from PC"), and that a jumper is installed on JP1. (see Figure 1). In this position, the serial port talks directly to the AI Motors and the board enters a bootloader mode in which it doesn't try to send commands to the motors.
4. Connect only one AI Motor to the controller board.
5. Install 6 AAA batteries in the battery holder, and plug the holder into the on-board jack. A small green LED near the serial connector will turn on. Another LED near the other side of the board will blink. The blinking LED indicates that the microcontroller is active, but is waiting before it puts any data on the serial line so that the AI Motor Tool can be used without conflicts.
6. To establish communications, click on the *Baud Scan* button. 57600 should now be displayed above it. Click on the *ID Scan* button. Scan ID should now show 0. Both are fine for the first motor.

7. Assuming you haven't changed the *Act ID* and *Act Mode* (on the left side of the GUI) from 0, wiggling the slider under the *Command Button* will cause to motor to move.
8. Disconnect the first motor and connect the second. Repeat the *Baud Scan* and *ID Scans* as in step 6 above. Results should be the same.
9. You need to set the ID of this motor to 1. Select 1 from the *Set ID* menu, then click on the *ID Set* button.
10. After selecting 1 for the Act ID, you can now control this second motor from the slider.

It's important to have only one motor connected when first initializing a motor and giving it an ID number. All motors are shipped with a default ID of 0 and having more than one motor with the same ID connected can disrupt communications in a very confusing way. Once you've given each motor a unique ID number, you can connect up to 31 motors to a single serial buss.

The SingleSetting panel also allows you to examine and change the various settings for the AI Motors. To change a setting, the *ID Scan* must agree with the ID of the motor you want to initialize. To change the baud rate, first use *Baud Set* and after that change the GUI's baud rate (upper left corner).

The following examples, including the robot demo, assume that right-hand motor is set to ID 0 and that the left-hand motor is set to ID 1, with other settings at the factory default.

1.2.3 Example 2: Manual Control

Once each motor has been initialized to a different ID, up to 31 motors can be connected to a single interface. The AI Motor Tool can be used to issue single commands from the leftmost part of the GUI screen. These commands are as follows.

- Act ID – selects which motor ID to command.
- Act Mode
    - 0-4 sets the speed. The numbering is reversed (0 is fastest, 4 is slowest)
    - 5, PositionRead, issues a command for the motor to return its current position (and load)
    - 6, "MotorDown", turns off the motor but leaves the motor's microprocessor active. This command should be issued when you want to manually move the motor to a particular position (otherwise the motor will fight to maintain the last instructed position).
    - 7, PowerDown, removes torque from the motor (as in Command 6) and puts the motor's microprocessor in power-conserving standby. It comes out of standby at the next command.
    - 8, "WheelActCCW", puts the motor in full rotation mode moving counter-clockwise.
    - 9, "WheelACTCW", puts the motor in full rotation mode moving clockwise.

NOTE: The Position Read mode also turns off motor torque. To get position and loading feedback without affecting the motor, just re-issue the last command.

The following walks you step by step through commanding a motor to a specified position, looking at motor current loading, reading the position of a motor, and controlling full-rotation.

1. Connect both motors to the controller board. This can be either directly to any of the connectors in Figure 1, or by daisy-chaining the motors.
2. At the far left of the AI Motor Tool GUI, use *Act ID* to select which motor you want to control. Select *Act Mode* of 0 to 4 for whatever speed you'd like.
3. Enter a value from 0 to 255 into *Command Value*. This is the position that will be commanded.
4. Click the *Command Send* button to issue the command. You can also use the slider to send out a rapid series of position commands. Experiment with a few positions and speeds.
5. You'll notice that each commands results in two additional values being displayed in *Return Value* at the bottom of the GUI. These values are in hexadecimal. For most commands, the first of each pair indicates motor current[1] and the second is the position. Both values are at the time the command was issued. So current will be 0 or very low, since the motor isn't doing any work while waiting for the command, and the position will be the prior target position. Add one of the mechanical attachments (like the geared wheel) to the side axel and apply some resistance while giving another position command. You'll see the current value go up. And that 7 kg*cm is pretty strong.
6. Select *Act Mode* of 6 and click *Command Send* to remove torque from one of the motors. Mechanically move the motor to a new position, then select *Act Mode* of 5 and click *Command Send*. The new position will be displayed in the *Return Value* panel (the Command Value won't be updated).
7. Select *Act Mode* of 7, select a *WheelSpdLevel* (here a higher number is faster), then click *Command Send*. The motor will rotate counter-clockwise until it's given a new command. If that command is another CCW or CCW rotation, the Return Values will be the number of rotations since the last command (rolling over after 255 decimal) and the current position.
8. While that motor continues to turn, you can issue position, full-rotation, and other commands to the other motor. Be sure to change the *Act ID* to that of the other motor.

Just remove the power supply plug from the controller board jack to turn everything off.

[1] Multiplying the current value, converted to decimal, times 18.5 gives the approximate motor draw in milliamps. Current draw is roughly proportional to the mechanical loading on the motor.

1.3 Running the Built-In Demo

The Evaluation kit is shipped with a ready to run demonstration program already burned into the microcontroller. In the demo, the robot is a photovore, approaching light and avoiding dark areas. The demo also tests all the populated circuitry. You can change this program, as described in the third part of this document.

Be careful not to overtighten the screws and nuts. Nylon hardware avoids the risk of electrical shorts and provides a little flexing, but its easier to strip the threads especially on small screws.

Figure 2 – Bottom with spacers

1.3.1 Build the Two-Wheel Robot

1. Connect the two AIMs together using two ¾" screws, two spacers, and two nuts, as shown in figure 2. This will be the bottom side of the 'bot. In this upside down position, the motor with ID #0 should be on the left and #1 on the right given the orientation of Figure 2. Leave the screws a little loose for now.
2. Flip the motors over and insert one connector assembly into each motor. Be sure each is fully inserted. It doesn't matter which gets the short cable and which gets the longer cable. The motor cables could be daisy-chained, but this would put a cable on the underside where it's likely to snag.
3. Build and attach two skids (tails) using the AIM parts, as shown in Figure 3. You can use the long metal screws and nuts in the AIM packages to firmly attach the skids, but they pretty much stay in place by themselves.

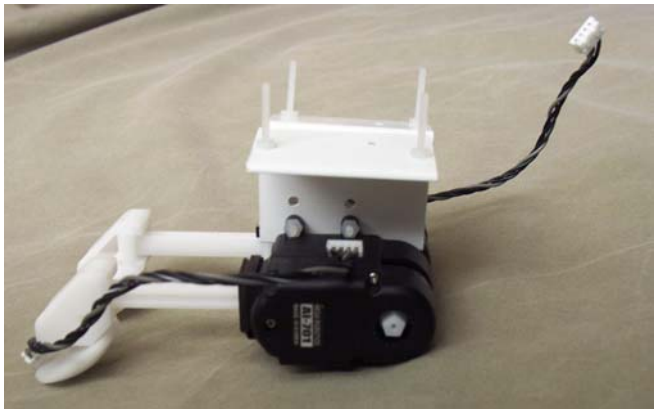


Figure 3 – Cables, Skids, and Brackets

4. Attach the two brackets using two ¾" screws and two nuts, also as shown in Figure 3, using the lower two holes for the AIMs. Using the upper two holes puts the controller board closer to the floor, but makes it *very* tough to get your fingers in to do the work. Leave the screws and nuts a little loose for now.
5. Attach two wheels and short metal screws from the AIM parts, as shown in Figure 4. You'll need to rotate each wheel to align it on the hexagonal AIM shaft before you can push it in. Install the ETEDS (Elastomeric Traction Enhancement Devices, aka rubber bands).

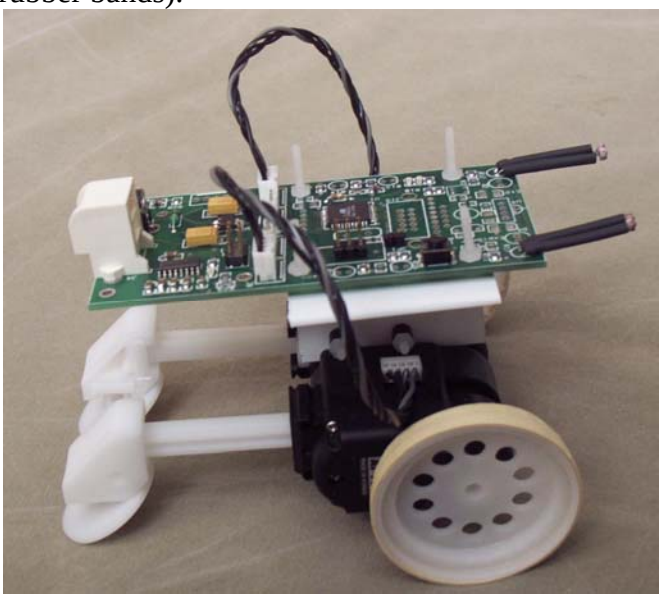


Figure 4 – Wheels and Controller Circuit Board

6. Attach the circuit board, also shown in Figure 4, with four ¼" screws, four nuts, and 4 washers. Place the washers between the circuit board and the mounting brackets. Leave the nuts a little loose.
7. After adjusting alignment and positioning, tighten all the screws and nuts. Be careful not to over-tighten; the nylon threads are easily damaged. Don't cause the circuit board to flex; leads on the bottom may put it at a slight angle. Attach the two AIM cables, noting the polarized connectors.

8. Insert 6 AAA batteries into the battery holder, observing polarity. Slide the battery holder between the tails and the circuit board with the cable on the side as the circuit board's power connector. Secure the battery back to the skits with the Velcro strap, as shown in Figure 5. Do not connect power to the board just yet.

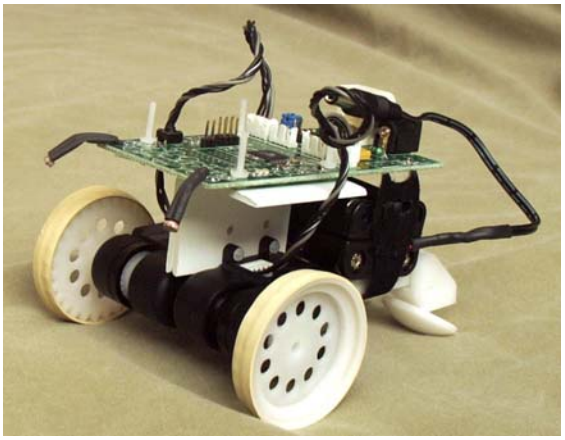


Figure 5 – Battery Pack and Light Sensor Pointing

9. The light sensors (CdS photo-resistors) should be pointing outward about 45^{o} and downward about 45^{o}.
10. Move the jumpers on J10 and J11 to the lower MCU position, as shown in Figure 1. If you leave the jumpers in the upper position, the motors remain connected to the RS232 serial interface and not the on-board microprocessor. Don't forget to remove the jumper on JP1. This allows the on-board processor to execute its built-in program.

1.3.2 Example 3: Run the Robot Demo

For your first run, you'll want to put the 'bot up on blocks so the wheels are off the ground and it doesn't go anywhere. An empty AI Motor box with the unused parts inside makes a good-enough stand.

1. Connect the battery pack to the microcontroller board.
2. As before, the power LED will turn on and another LED will blink. To begin the robot demo, completely cover the right light sensor. The sensor is fairly sensitive even from the side, so pinching it between a thumb and finger is a good way to completely cover it. The board will then go through a brief blinking-LED sequence. The board then enters an AIM position and force read-out demo.
    a. *Slowly* rotate the left wheel, and right wheel will copy the motion. The MCU is reading the position of the left wheel and sending that position as a command to the right wheel. The MCU is also monitoring the load current of the right motor.
    b. If you twist the right motor (hard) or rotate the left motor quickly, the program will reverse the demo…
    c. The right wheel position is read and the left wheel is slaved to it.
    d. After 5 seconds, the program goes on to the next step of the demo.
3. The board then enters an AIM full-rotation mode and "odometry" demo, executing a sequence of commands. The MCU gives the motors full-rotation

commands. Since motor speed varies somewhat with battery voltage, the MCU frequently repeats the commands to get updated rotation count and position from the AIMs. Basing the motion sequence on motor feedback gives greater accuracy than a sequence based on fixed times. The 90^{o} turn uses elapsed time to illustrate the traditional method. The turns will become smaller as battery voltage falls. Robustly handling both the rotation counter (which counts in the opposite direction as the position count!) and the position, with unsigned comparisons gets a bit messy. So we'll be including a routine for this in the upcoming AIM addition to the library. If the 'bot is on the floor, it will move in a square roughly 1 foot on a side. It'll then enter the last phase of the demo.

4. The robot will then go into a light-tracking mode. To get good light-following action you may need to adjust the program's coefficients as described in Section 3.
    a. If ambient light is bright enough, the 'bot will move slowly forward, turning in the direction of the brightest light. Otherwise it'll just sit there.
    b. You can direct the robot around with a flashlight. Or by shading one of the light sensors.
    c. If you cover the right-side light sensor, the MPU will shut down the AIMs and end the demo.

Pressing the reset pushbutton restarts the demo sequence.

The source code for this demo is included on the CD-ROM. It exercises most of the AIM functions (other than initialization) as well as the control-board sensors, and can serve as a good starting point for your own programming, as covered in the Part 3 of this manual. Subsequent programming will erase this example. It can be re-loaded from *c:\Code\AVRC\avrlib\examples\AIMintro\AIMintro.c* once you've done the software installation in Section 3.

Part 2: Specifications

This section summarizes specifications for the AI Motors, and provides the full specifications for the controller board and demo robot. The next section will cover custom programming the controller using C.

2.1 AI Motors

This evaluation kit comes with two AI-701 AI Motors. For the full specifications, features, and description of use, see *The AI Motor-701 Manual*. The '701 and '1001 motors are interchangeable, the primary difference being that the '1001 uses a higher gear ratio to achieve higher torque at lower speed.

2.1.1 Summary of Specifications

- Stall torque 7 kg*cm @ 9.5 V
- Maximum speed 82 rpm @ 9.5 V
- Weight 40 g
- Size 51.6 x 34.3 x 37.1 mm
- Power
    - Maximum 650 mA @ 5 VDC, 1000 mA @ 10 VDC
    - Typical 100—200 mA
    - Minimum 12 mA
- Positioning range and resolution
    - Low resolution mode – 0 to 333.3 degrees, 1.31 degrees resolution
    - High resolution mode – 0 to 166.6 degrees, 0.654 degrees resolution
- Speed control
    - Position mode – 16 steps
    - Full rotation mode – 5 steps
- Up to 31 motors on a single serial bus (asynchronous serial at TTL levels)
- Baud rate 2,400 to 460,800 bps
- Other settable parameters
    - Movement bounds
    - Over-current threshold
    - P- and D-gain

2.1.2 Summary of Protocol

The following table summarizes the AI Motor protocol. The most used command packets are 4 bytes long. Initilization commands are 6 bytes long. All command packets except synchronous move are acknowledged by 2 bytes from the motor. Here we use conventional C notation for packing data into the command bytes. For some commands certain bit-fields in command bytes 3, 4, and 5 can have arbitrary values, but that is ignored in this simplified table. A blank in the table indicates that no byte is sent (4-byte commands).

There is a special command protocol to move several motors simultaneously. The motor IDs must be sequential and start at 0.

- Cmd Byte1 – 0xFF
- Cmd Byte2 – (spd) << 5 | 0x1F
- Cmd Byte3 – last ID
- Cmd Byte4 – position for ID=0
- Cmd Byte5 – position for ID=1
- …
- Cmd ByteN – position for last ID
- Cmd ByteN+1 – checksum = exclusive-or of all positions anded with 0x7F

There are no acknowledgement bytes for the synchronous position send command.

Name	Cmd Byte1	Cmd Byte2	Cmd Byte3	Cmd Byte4	Cmd Byte5	Cmd Byte6	Ack Byte1	Ack Byte2
Position Send	0xFF	(spd[2])<<5 | ID	Position	Chksum1[3]			Current	Position
Position Read	0xFF	0xA0 | ID	0x00	Chksum1			Current	Position
Act Down[4]	0xFF	0xC0 | ID	0x10	Chksum1			Cmd Byte3	Position
Power Down	0xFF	0xDF	0x20	Chksum1			arbitrary	arbitrary
Full Rotation	0xFF	0xC0 | ID	(dir[5]) << 4 | (spr[6])	Chksum1			Net rotation count	Position
Synch-ronous Move	See text	See text	See text	See text	See text	See text	None	None
Set Baud	0xFF	0xE0 | ID	0x08	rate[7]	Cmd Byte4	Chksum2[8]	New baud	New baud
Set Gains	0xFF	0xE0 | ID	0x09	Pgain	Dgain	Chksum2	New Pgain	New Dgain
Set ID	0xFF	0xE0 | ID	0x0A	New ID	New ID	Chksum2	New ID	New ID
Read Gains	0xFF	0xE0 | ID	0x0C	0x00	0x00	Chksum2	Pgain	Dgain
Set Resolution	0xFF	0xE0 | ID	0x0D	Resol, 0 or 1	Cmd Byte4	Chksum2	New resol	New resol
Read Resolution	0xFF	0xE0 | ID	0x0E	0x00	0x00	Chksum2	0/1 (resol)	0/1 (resol)
Set over-current threshold	0xFF	0xE0 | ID	0x0F	Threshold mA/18.5	Cmd Byte4	Chksum2	New Thresh.	New Thresh.
Read over-current threshold	0xFF	0xE0 | ID	0x10	0x00	0x00	Chksum2	Thresh.	Thresh.
Set Bounds	0xFF	0xE0 | ID	0x11	Min pos	Max pos	Chksum2	New min pos	New max pos
Read Bounds	0xFF	0xE0 | ID	0x12	0x00	0x00	Chksum2	Min pos	Max pos

Table 1 – AI Motor Commands

2.2 Controller Board

The AI Motor Introductory kit includes a printed circuit board combining RS232—TTL level converter, power regulator, microcontroller, and sensors. On the introductory kit, most sensor sites are unpopulated.

[2] "spd" is 0 to 4 indicating speed; 0 is fastest
[3] Checksum1 = Byte2 ^ Byte3 & 0x7F
[4] Turn off motor torque.
[5] "dir" is direction, 3=counter clockwise, 4=clockwise.
[6] "spr" 0 to 15 indicating rotation speed; 0 is stop, 15 is fastest
[7] See AI-701 User's manual
[8] Chksum2 = (Byte2 ^ Byte3 ^ Byte4 ^ Byte5) & 0x7F

2.2.1 Mechanical

Figure 1 shows the connectors, jumpers, and mounting hole locations. Outer dimensions are 2.125 x 4.00 inches.

2.2.2 Power

External power is provided by battery or UL listed power converter. Input power must be 6.5 to 10 V DC. Input voltage is directly routed to the AI Motor connectors, and regulated to 5 V DC for on-board circuitry. Current draw is approximately 50 mA, excluding the motors. The power connector is a 2.1 mm jack, center positive.

2.2.3 External I/O

As populated for the AI Motor Introductory Kit, the controller board provides the following external connectors.

- **RJ11-6P4C** female connector, with true RS232 voltage levels.
- **4 2mm headers** for AI Motors**.**

As shipped, the serial adapter cable provides the following connections.

DB9 (female)	RJ11-6P4C (male)
1	
2 RXD	2 (data from AIM to PC) (red)
3 TXD	3 (data from PC to AIM) (green)
4	
5 GND	1 (black)
6	
7	
8	
9	

Table 2 – Serial Cable Pinout

Pin 4 (yellow) of the RJ11-6P4C connector is not connected. With the jumper changes described later, it can be used to receive a handshaking signal from the host.

Pinout for the AIM connectors is as follows:

1 Ground
2 AIM RX (data from PC to AIM)
3 AIM TX (data from AIM to PC)
4 Supply voltage

All AIM connectors are the same serial buss. Multiple AIMs can be daisy chained from a single connector; 4 connectors are provided for convenience.

2.2.4 On-Board Functions

As populated for the AI Motor Introductory Kit, the controller board includes the following functions.

- **RS232 conversion to TTL** levels for communication with AI Motors.
- **ATmega 16** microcontroller. 8 MHz, 16 Kbytes program Flash, 1 Kbyte data SRAM (internal oscillator)
- **Reset pushbutton**
- **Power LED**
- **2 user-programmable LEDs**
- **2 light sensors**

Serial communication can be jumpered as shown in Figure 1. Placing a jumper on JP1 and pressing reset will activate the program bootloader. Otherwise power-on or reset will restart execution of the last program loaded.

2.2.5 Software Included

- AI Motor Tool
- Atmel IDE with assembler, *debugger, and simulator*
- Open-source C compiler (GCC for the AVR)
- Open-source drivers, device library, and example code (AVRLIB)

The following library routines have been configured for and tested on the AIM Introductory board. All include example code.

- a2d
- rprintf
- uart

2.3 Two-Wheeled Robot

The two AI Motors and the controller board are combined with a minimum of additional parts to produce a demonstration robot.

- Maximum speed – about 8” (0.2 m) per second
- Total power draw – 2 to 3 W when active

Sensing and processing capabilities are covered in the controller board specification, and the motors in the AI Motor specification and User’s Manual.

Part 3: Programming

This section presents instructions on loading the development software and a walk-through of the programming cycle. We then cover the hardware features, mapping them back to the ATmega registers. Finally, we introduce the AVRLIB functions as they relate to the AIM Introductory kit.

This document assumes familiarity with C and some familiarity with AVR microcontrollers and embedded system programming.

3.1. Software Installation

The controller board uses a Windows PC as the development host. All software is available as free downloads from the internet. Garage Technologies provides a distribution CD ROM for your convenience. This section summarizes a standard installation. To date the only anomalies we've found are on systems where Cygwin has already been installed. The GCC compiler package installs portions of Cygwin to support make files and Linux-like commands. The GCC installation procedure re-installs Cygwin components with a possibly different directory structure and different .dll file versions. So if you've already installed Cygwin you're likely to have to patch your Cygwin installation to get it working again.

3.1.1. Install AVR Studio 4.1

First, install the Atmel IDE for downloading programs into AVR processor chips. The Studio also incorporates other functions, such as debugger and simulator that aren't covered in this manual. It can also be downloaded from www.atmel.com.

1. For WindowsNT/2000/XP you must be logged in with administrator rights, as new drivers are installed.
2. Put the Garage Technologies CD-ROM into the drive and navigate into that drive.
3. Go into the *CdevelopmentToolChain* directory and click on the icon for *aStudio4b356*
4. Follow the on-screen instructions. We recommend letting everything default, including
    a. Letting the directory default to *c:\Program Files\Atmel\AVR Tools*
    b. No need for USB tools
5. Upon finishing let the installer reboot your machine if requested

This will install *Atmel AVR* Tools to your programs menu.

3.1.2. Install GCC

Next, install WinAVR. This is a build of the GNU C Compiler specific to run on Windows systems and compile code for AVR microcontrollers. It can also be downloaded from www.sourceforge.net/projects/winavr.

1. Put the Garage Technologies CD-ROM into the drive and navigate into that drive.
2. Go into the *CdevelopmentToolChain* directory and click on the icon for *WinAVR-20040720-install*
3. Follow the on-screen instructions. We recommend letting everything default, including
    a. Letting the directory default to c:\WinAVR
    b. Leaving the options all clicked
6. The installer will bring up a window with Programmer's Notepad 2 and information on the GCC installation. If you wish, review the detailed information and close the window.
7. Upon finishing, let the installer reboot your machine if requested.

This will install several icons on your desktop: *AVR Insight, AVR libc manual, GNU manuals online, Programmer's Notepad, README,* and *tkInfo*. We'll only be using the *Programmer's Notepad* in this manual.

3.1.3. Install Procyon Library

This open-source GPL software by Pascal Stang is a very useful library of functions for AVR processors. The CD ROM contains those routines we've tested on the AIM Intro board. We'll incorporate additional library routines as we configure and test them for Garage Technologies products. There's no automatic install software for this library. The following instructions include default file folder names and locations and are recommended for compatibility with various examples.

1. Use the Windows *My Computer* icon to navigate to *c:* (root of the c disk drive).
2. Create a new folder called *Code*.
3. Enter that folder and create a new folder named *AVRC*.
4. Copy the CD ROM folder *avrlib* into *c:\Code\AVRC*. The full path then becomes *c:\Code\AVRC\avrlib\...*.
5. You now need to create an environment variable saying where the library is. The procedure for this depends on which version of Windows you're using.
    a. For Windows 2000/XP,
        i. Go to the *Control Panel*
        ii. Double-click *System* and click the *Advanced* tab
        iii. Click the *Environment Variables* button
        iv. Use the *New* button to add an environment variable for the user or the full system
        v. Enter *AVRLIB* as the environment variable name in the *Variable Name* field
        vi. Enter *c:\Code\AVRC\avrlib* as the value in the *Variable Value* field
        vii. Click *OK* then *OK* again to set the environmental variable, then another OK.
        viii. Then manually reboot.
    b. For Windows95/98 or Windows 3.1, there's not an easy built-in way for setting environment variables. One way is to use *Wordpad* to edit the file *c:\autoexec.bat*. Add the following line to the end of the file: *SET AVRLIB =c:\Code\AVRC\avrlib* If you don't want to risk editing your

autoexec.bat file (a major mess-up can keep your PC from booting!), you can enter the same SET command file from a DOS window each time you open a new DOS window for compiling C code.

c. For Windows NT,
    i. Go to the *Control Panel*
    ii. Double-click *System* and click on the *Environment* tab
    iii. Enter *AVRLIB* as the environment variable name in the *Variable* box
    iv. Enter *c:\Code\AVRC\avrlib* as the value in the *Value* box
    v. Click *Set* to set the environmental variable

3.1.4. Test Your Installation

1. AVR Studio doesn't load a desktop icon. Navigate to the programs menu, *Atmel AVR Tools → AVR Studio 4*. If the tool comes up the installation should be fine. See *AVR Tools Online Help* for introduction and assistance.
2. The winAVR build of GCC uses parts of Cygwin to recreate many Unix / Linux commands within a DOS window.
    a. Open a DOS command window, then move to the kit's example directory with *cd c:\Code\AVRC\avrlib\examples\AIMintro*.
    b. Enter *make clean*. This will natter a bit and clean up compilation files but not report any errors.
    c. Enter *make*. This will re-compile the example C source code. There'll be a lot of nattering, but there should be no errors.

That was a lot of effort, and there's a bit more to learn on how to edit, compile, and download a C program. But once you've gotten the hang of it, the Procyon library and examples make programming surprisingly quick.

3.2. An Edit-Compile-Program-Run Iteration

In this walk-through we'll make a minor modification to the demo program to illustrate the edit-compile-load-run cycle.

Editing Source Code

1. Open the *Programmer's Notepad*. You can also use your favorite text editor, but be sure to save as plain text. Programmer's Notepad "knows" C syntax and provides helpful text coloring and other features.
2. From the Notepad, open *c:\Code\AVRC\avrlib\examples\AIMintro\AIMintro.c*.
3. In the "position & force readout demo" portion of the code, change *while(curr<3)* to *while(curr<5)*. This will make it noticeably more difficult to get the mastering to switch from the left to the right motor.
4. Save your modified file

Compile the Code

1. Open a DOS window (run cmd.exe)
2. cd to *c:\Code\AVRC\avrlib\examples\AIMintro*.

3. Enter *make*. This calls a make utility that compiles AIMintro.c and any needed libraries, and links them into a downloadable file. You'll see lots of nattering, but towards the last few lines you should see some info about a. AIMintro.elf file of a little over 5 KBytes (the total size of your compiled code), and the message "Errors: none"

Load and Run the Code Using the Bootloader

1. Connect a serial cable between your PC's serial port and the AIM controller board.
2. Open AVR Studio, cancel the *Create New Project* window pop-up.
3. Install a jumper block on JP1. This tells the MCU to execute the bootloader rather than the previously loaded application program.
4. Press the reset button on the AIM controller circuit board. The left user LED should be on to indicate that you're in the bootloader.
5. In AVR Studio, select *Tools→Avr Prog*. It should detect the ATmega16 MCU and ask you to browse to the hex file you want to load in the flash. Browse to *c:\Code\AVRC\avrlib\examples\AIMintro* and select *AIMintro.hex* and the bootloader should load the hex file into the MCUs Flash memory.
6. Remove JP1 and press the reset button again. Your new program will begin to execute.

If you get the message "*no support board found*," re-check your jumpers, connections, and board power. Also check to be sure there's no Atmel Studio programming window open and hiding elsewhere.

You can also program the controller board, including loading a different bootloader program, using a serial- or parallel-port programming pod as described in the Atmel documentation.

3.3 Hardware Interfaces

Here we cover the microcontroller lines used for on-board features and off-board interfacing. It's probably more clear and safer to describe these features and input/outputs in terms of the ATmega ports. This also gets us one step closer to the details of programming the various capabilities.

3.3.1 Port A: Digital I/O or Analog Input

Most of the Port A pins go to on-board voltage dividers for resistance-based sensors such as CdS light sensors and thermistor temperature sensors, as shown in the schematic (see the Appendix)

Port A	On-Board	Use
PA0 / ADC0	R17	Right light sensor
PA1 / ADC1	R16	
PA2 / ADC2	R15	
PA3 / ADC3	J13	
PA4 / ADC4	J12	
PA5 / ADC5	R14	
PA6 / ADC6	R13	
PA7 / ADC7	R12	Left light sensor

Table 3 -- Port A

Analog reference is +5 V Vcc, isolated by an RC filter, so inputs are 0-5V. The unpopulated sites R13—R16 have .1" spaced pads and can be used for jumpers to be read by software.

3.3.2 Port B: SPI

With direct connections to the microcontroller, it's important be sure the pin doesn't see too high a voltage (5.0 V DC max.) or have to source or sink too much current (20 mA max.).

Port B	SPI Header
PB0 / T0	
PB1 / T1	
PB2 / AIN0	
PB3 / AIN1	
PB4 / SS	
PB5 / MOSI	Pin 4
PB6 / MISO	Pin 1
PB7 / SCK	Pin 3

Table 4 -- Port B

The SPI header is configured to be compatible with in-system programmers, in addition to general SPI use. It can also be used as 3 general-purpose digital I/O.

Pin 1 – MISO (PB6)	Pin 2 -- +5 V Vcc
Pin 3 – SCK (PB7)	Pin 4 – MOSI (PB5)
Pin 5 -- *Reset	Pin 6 -- Ground

Table 5 -- SPI Header Pinout

3.3.3 Port C: LEDs

Port C	On-Board
PC0 / SCL	
PC1 / SDA	
PC2 / TCK	J14
PC3 / TMS	
PC4 / TDO	
PC5 / TDI	
PC6 / *TOSC1*	User LED
PC7 / *TOSC2*	User LED

Table 6 -- Port C

Port D: Serial RS232

Port D	On-Board
PD0 / RXD	Serial I/O
PD1 / TXD	Serial I/O
PD2 / *INT0*	Jumper JP1
PD3 / *INT1*	
PD4 / *OC1B*	
PD5 / *OC1A*	
PD6 / *ICP*	
PD7 / *OC2*	

Table 7 -- Port D

Jumper JP1 is used to activate the bootstrap loader.

3.4 Software Development

Here we discuss the details of programming specific features on the controller board. We're not covering the principles and techniques of embedded system programming. There's additional comments and documentation in the example and library code itself. If a library element has been configured and tested for the board, this will be noted in the revision list comments within the code. The library and documentation are continually updated, so check *www.Garage-Technologies.com* for the latest. Here we're emphasizing the basics of port input-output. Accessing registers inside the ATmega is similar, and together these form the foundation of programming the special hardware functions of an embedded system.

3.4.1 Port I/O

Input/output pins on the ATmega are grouped into 8-bit ports. On the ATmega8535, 16, and 32, these are ports A-D for a total of 32 lines. These are used by both on-board resources and the input-output headers as discussed above. Each port has 3 corresponding registers within the ATmega. One, DDRx (where x is A, B, C, or D) is used to determine whether each pin in an input or output. The other two, PORTx and PINx, are used for output and input respectively. Input, output, and special functions (such as timers and counters) can be mixed on the same port. For example, the UART and serial interface are assigned by hardware to 2 pins of Port D. This complicates programming a bit. Libraries, such as AVRLIB, are especially helpful in hiding this low-level "bit-banging" allowing the programmer to concentrate on the end application.

At the lowest level, DDRx, PORTx, and PINx can be used in C assignment statements with 8-bit variables or constants to set direction, output, or input respectively. This sets, outputs, or reads all 8 bits. The C functions inb(port, value) and outb(port, value) are preferred, however. Individual bits can be set by sbi(port, pin#) and cleared by cbi(port, pin#) without disturbing any of the other pins.

Comments in the code *…/avrlib/examples/basicio/basiciotest.c* include a more lengthy discussion of I/O pins and port programming.

Specific to the AIM Introductory kit board, 2 LEDs are connected to PC6..7 (Port C, pins 6 and 7). Like most microprocessors, AVR outputs are better at sinking current rather than sourcing current, so the LEDs are pulled up to +5V through a 1 K resistor. Setting the output to 0 grounds the other LED pin and turns the LED on. Jumper JP1 is connected to PD2. By enabling the PD2 pullup (PORTD = 00000100 even though it's input), this jumper is a software readable input. There's no debouncing, so some software may require extra checking of the switch level.

3.4.2 Analog Input

Analog input uses the Port A pins, connected to the internal A/D (analog to digital) converter. The ATmega A/D converter and related circuitry is controlled by setting and monitoring numerous internal registers. AVRLIB does most of the low-level work for you in a library of functions called a2d. The user must call a2dInit() to initialize the converter, plus a few routines to set options such as conversion time (related to accuracy). It's also necessary to be sure that any Port A pins used for analog input have their corresponding DDRA pin set to 0 (input) and their corresponding PORTA pin also set to 0 (turning off the internal pull-up resistor). You can then call a2dconvert8bit() or the corresponding 10-bit call go get the current value of a particular input.

On the AIM Introductory board, PA0..2 and PA5..7 are connected to pads for resistance type sensors, with a 1 K pullup to +5 V forming a voltage divider. PA0 and PA7 are populated with a Cds light sensor. The resistor values are such that at typical room

conditions the division ratio is a little higher than 2. A/D readings returned will thereby be very roughly 100 for 8-bit conversions. Software can convert these raw readings to calibrated engineering units (degrees F, degrees C, lux, lumens, etc.). The *AIMintro.c* code illustrates this, and the *../avrlib/examples/a2d/a2d.c* code provides additional comments and code examples.

3.4.3 UART (Serial Port)

AVRLIB supports serial-port communications using the on-chip UART and an on-board RS232 level converter. */avrlib/uart* contains code and comments on the lower-level UART library functions. Most people will use the higher-level rprintf library. The rprintf library provides functions to initialize the UART, set output to the serial port, and "print" character strings. It's the equivalent of printf of the non-embedded C world. Detailed comments and examples are in *…/avrlib/examples/rprintf*.

Appendix 1 – Example Code

The following code is a slightly reduced version of the demo code pre-loaded into the controller. The full version is included in the examples directory on the CD ROM. This version shows the AI Motor functions, with fewer comments, and omits the fixed-path demo. Note that a "standard library" and initialization for the AI Motor running commands is less than 2 pages of code and each demo is less than 1 page.

```
//*********************************************************************
// File Name    : AIMintrolite.c
//
// Shortented version of AIMintro.c as program listing example
// of using AIM motors and the AIM Intro Kit
// GPL
// 17-Jan-2005 Garage Technologies, Inc.
//*********************************************************************

//----- Include Files ---------------------------------------------------
#include <avr/io.h>             // include I/O definitions (port names, pin names, etc)
#include <avr/signal.h>         // include "signal" names (interrupt names)
#include <avr/interrupt.h>      // include interrupt support

#include "global.h"             // include our global settings
#include "uart.h"               // include uart function library
#include "rprintf.h"            // include printf function library
#include "timer.h"              // include timer function library (timing, PWM, etc)
#include "a2d.h"                // include A/D converter function library

#define abs(A)(((A)>0) ? (A) : (0-(A)))

//*****************************AIM "Library"*******************************
// should be enums...
#define aPOS        1       // AIM send position
#define aPOSr       2       // AIM read  position
#define aDWN        3       // AIM act-down (remove torque)
#define aPDN        4       // AIM power-down
#define aROT        5       // AIM full-rotation
#define CCW         3       // AIM full-rotation direction
#define CW          4       // AIM full-rotation direction
#define LEFT        1       // left AIM
#define RIGHT       0       // right AIM

//transmit 4-byte AIM command
void aimcmd4(u08 b2, u08 b3)
{
        rprintfChar(0xFF);
```

```
        rprintfChar(b2);
        rprintfChar(b3);
        rprintfChar((b2^b3)&0x7F);
}

// get  from AI motor
// NOT CHECKING FOR UART BUFFER EMPTY
u16 aimAck(void)
{
        u08 b1=0,b2=0;

        timerPause(1); // give motor time to respond
        uartReceiveByte(&b1);
        uartReceiveByte(&b2);
        return((b1<<8) | b2);
}


u08 hiByte(u16 x)
{
        return (x>>8);
}

u08 loByte(u16 x)
{
        return (x&0x00FF);
}

// send a command to AI motor
u16 aimCmd(u08 cmd, u08 id, u08 pos, u08 spd)
{
        u16 readback = 0x0000;

        readback = 0x0000;
        switch(cmd)
        {
                case aPOS:      // send a positioning command
                        aimcmd4((spd<<5)|id,pos);
                        readback = aimAck();
                        break;
                case aPOSr:     // command read-position
                        aimcmd4(0xA0|id,0x00);
                        readback = aimAck();
                        break;
                case aDWN:    // act-down
                        aimcmd4(0xC0|id,0x10);
```

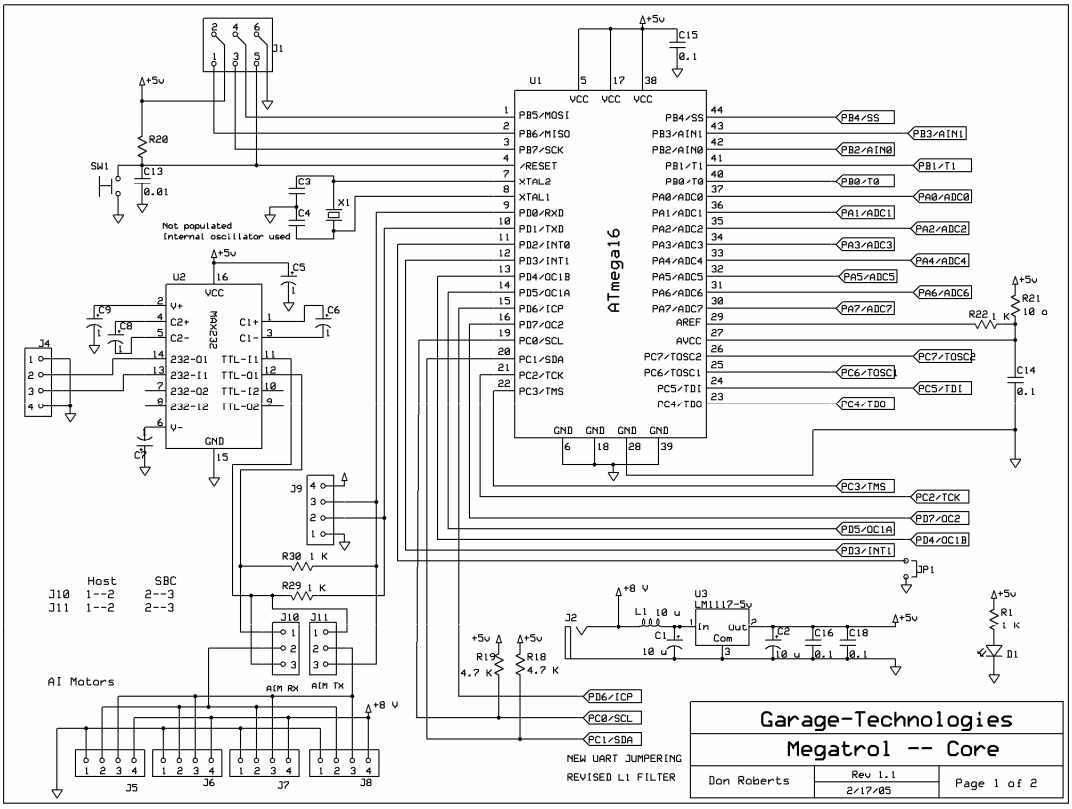
```
                readback = aimAck();
                break;
        case aPDN:     // power-down
                aimcmd4(0xDF, 0x20);
                readback = aimAck();
                break;
        case aROT:     // full rotation modes
                aimcmd4(0xC0|id, (pos<<4)|spd);
                readback = aimAck();
                break;
        default:
                break;
    }

    return readback;
}

//***** Begin Code ************************************************************
int main(void)
{
    u08 i=0;          // unsigned 8 bit
    u08 curr,pos;
    u16 myack;
    u08 leftLight, rightLight;

    // ----- Initialization ---------------------------------------------

    // initialize our libraries
    // initialize the UART (serial port)
    uartInit();
    // make all rprintf statements use uart for output until told otherwise
    rprintfInit(uartSendByte);
    uartSetBaudRate(57600);       // AI motor default
    // initialize the timer system
    timerInit();
    // turn on and initialize A/D converter
    a2dInit();
    // Make the analog port input, be sure pull-ups are turned off
    DDRA = 0x00;
    PORTA = 0x00;
    // set a2d prescaler (speed and accuracy) and select voltage reference
    a2dSetPrescaler(ADC_PRESCALE_DIV32);
    a2dSetReference(ADC_REFERENCE_AVCC);
```

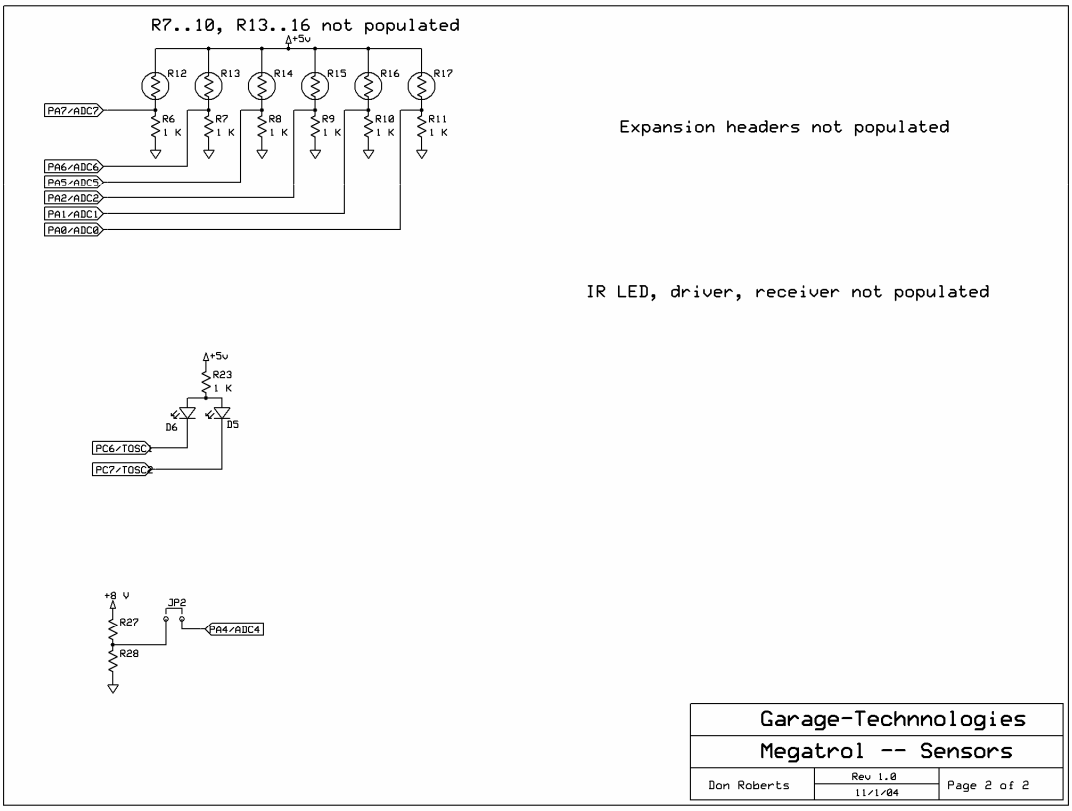
```
// ------------------- position & force readout demo -------------------
        // move both motors to middle
        aimCmd(aPOS,LEFT,128,1);
        aimCmd(aPOS,RIGHT,128,1);
        timerPause(1000);       // wait a second
        // remove torque from left motor
        aimCmd(aDWN,LEFT,0,0);
        curr = 0;
        // left motor as sensor
        while(curr<3)  // while right motor not loaded
        {
                // get position of left motor
                myack = aimCmd(aPOSr,LEFT,0,0); // also removes torque
                pos = loByte(myack);
                //send position to right motor
                aimCmd(aPOS,RIGHT,pos,1);
                timerPause(150); // let left motor get to position (with reduced current)
                // get current of right motor
                myack = aimCmd(aPOS,RIGHT,pos,1);
                curr = hiByte(myack);
        }
        // right motor as sensor
        for(i=0;i<50;i++)
        {
                // get position of right motor
                myack = aimCmd(aPOSr,RIGHT,0,0);
                pos = loByte(myack);
                // send position to left motor
                aimCmd(aPOS,LEFT,pos,1);
                timerPause(100);
        }
```

```
	// ------------------- photovore demo -------------------
	rightLight = a2dConvert8bit(0);
	while(rightLight>=20)		// covering right sensor will end demo
	{
		timerPause(100);
		leftLight = a2dConvert8bit(7);
		rightLight = a2dConvert8bit(0);
		if ((leftLight+rightLight)>250)
		{
			// bright enough -- stay put
			aimCmd(aROT,RIGHT,CW,0);
			aimCmd(aROT,LEFT,CCW,0);
		}
		else if (abs(leftLight-rightLight)<=10)
		{
			// move forward
			aimCmd(aROT,RIGHT,CW,6);
			aimCmd(aROT,LEFT,CCW,6);
		}
		else if (leftLight > rightLight)
		{
			// turn left
			aimCmd(aROT,RIGHT,CW,6);
			aimCmd(aROT,LEFT,CCW,0);	// by stopping left motor
		}
		else
		{
			// turn right
			aimCmd(aROT,RIGHT,CW,0);	// by stopping right motor
			aimCmd(aROT,LEFT,CCW,6);
		}
	}


	// ------------------ shut-down motors -------------------
	aimCmd(aPDN,0,0,0);


	return 0;
}
```

Appendix 2 – Schematics

The following schematics show the circuitry populated on the Introductory Kit controller board. If there's sufficient interest, we'll publish some application notes for using the IR LEDs and expansion headers.




R7..10, R13..16 not populated
+5v
R12
R13
R14
R15
R16
R17
PA7/ADC7
R6
1 K
R7
1 K
R8
1 K
R9
1 K
R10
1 K
R11
1 K
PA6/ADC6
PA5/ADC5
PA2/ADC2
PA1/ADC1
PA0/ADC0
Expansion headers not populated
IR LED, driver, receiver not populated
+5v
R23
1 K
D6
D5
PC6/TOSC1
PC7/TOSC2
+8 V
JP2
R27
R28
PA4/ADC4
Garage-Technnologies
Megatrol -- Sensors
Don Roberts
Rev 1.0
11/1/04
Page 2 of 2